IN THE SPECIFICATION

[0034] The present invention is directed to a method and apparatus for implementing a recent entry queue which complements a Branch Target Buffer (BTB) ~~BTB table~~ 100 as shown generally in FIG. 1. ~~FIG. 4 shows an example of a BTB table recent entry queue with compare logic 400. In FIG. 4 a set of data lines 410 supply data to a first set of recent entry queue registers 420 for a first entry and to a second set of recent entry queue registers 421 for a second entry. The registers 420 and 421 store branch tag, target address and valid data. The data in the first set of registers 420 is compared by a first compare equal comparator 430 with the data being supplied by data in lines 410. The data in the second set of registers 421 is compared by a second compare equal comparator 431 with the data being supplied by data in lines 410. When the compare equal comparators detect equality an output is supplied from either or both of them to the OR gate 440 which supplies a block write signal on line 450.~~ Through the usage of a BTB ~~table~~ recent entry queue ~~with compare logic~~ 400, as shown in FIG. 4, three benefits are acquired as follows:

1) Removal of majority of scenarios that can cause duplicate entries in ~~a~~ the BTB. ~~table 100 of FIG. 1 , BTB table 610 in FIG. 6.~~

2) The ability to semi-synchronize the asynchronous interface between branch prediction and decode ~~ing~~ when the latency of the branch detection via the BTB table initially places the BTB ~~table~~ behind the decode ~~ing~~ of a pipeline when the pipeline is starting up from a cold start or after a branch ~~which was~~ wrong.

3) For frequently accessed branches, the ability to access them in fewer cycles thereby improving the throughput of the branch prediction logic which in turn improves the overall throughput of the given microprocessor pipeline 300 of FIG. 3, with ~~the~~ decode ~~ing stage~~ 310, ~~cycle cache~~ address calculation ~~stage~~ 320, cache access ~~stage~~ 330, register access ~~stage~~ 340, execute and branch resolution ~~stage~~ 350, and register writeback ~~stage~~ 360.

[0036] When a branch is not predicted, a surprise branch 710, (FIG. 7) may be encountered, and it is to be written into the BTB ~~table~~ 100 in FIG. 1, and ~~the BTB table~~ 610 in FIG. 6 such that it can be predicted in respect to the next occurrence, upon ~~branch~~ resolution ~~stage~~ 350 of ~~that~~ branch at the execute time frame that the target of the branch and the direction of the branch resolution are known. It has been standard to use the known information at this time frame, per example, and write the branch into the BTB ~~table~~ 610. A branch can be a surprise branch 710 for one of two reasons: it was not in the queue, or it was in the queue but it was not found in time. In the latter case, the branch should not ideally be added into the BTB ~~table~~ 610 again, as doing so would most likely replace some other good entry which is different from the duplicate entry that is to be written in to the BTB ~~table~~ 610. Through the usage~~-timeline for write queue access and BTB writing timeline~~ 500 in FIG. 5 of a ~~BTB~~ recent entry queue 400, ~~in FIG. 4, or a BTB recent entry queue~~ 620 in FIG. 6, whenever a ~~new data~~ in entry 140 in FIG. 1 is to be written ~~during the send data to queue interval~~ 510 into the BTB ~~table~~ 610, it is first compared ~~with first and second compare equal comparators~~ 430, 431, ~~entry in the recent entry queue decision block~~ 720 ~~in FIG. 7~~ to the ~~branch tag~~ entries 420, 421 within the recent entry queue. If it matches ~~as indicated by an output from OR gate~~ 440 ~~in FIG. 4~~, ~~during the check for duplicate entry interval~~ 520~~-in FIG. 5,~~ one of the entries in the recent entry queue 620 then the ~~data in~~ entry 140, ~~or data in entry~~ 410 ~~in FIG. 4~~ is blocked ~~by block write~~ 450 ~~in FIG. 4~~ from being written into the BTB ~~table~~ 610 as it already exists somewhere within the BTB ~~table~~ 610. If the entry is not located within the recent entry queue ~~registers~~ 420, 421, 620 then the entry is written ~~during the write BTB write queue interval~~ 530 into both the BTB ~~table~~ 610 and ~~into~~ the recent entry queue 620. The recent entry queue ~~620~~ works in a first ~~data~~ in 410, ~~registers~~ 420--first out ~~registers~~ 421 (FIFO) queue structure. When a new entry is placed into the queue, the oldest entry in the queue is moved out to make room for the newest entry. Should an entry in the BTB ~~table~~ 610 be required to be invalidated for any reason, the recent entry queue ~~registers~~ 420, ~~and registers~~ 421 must be checked to determine if the entry is also contained within it. If the entry is in the recent entry queue ~~registers~~ 420, ~~registers~~ 421 and the entry is being invalidated in the BTB ~~table~~ 610, then the entry must also be invalidated in the recent entry queue ~~registers~~ 420, ~~registers~~ 421.

[0037] ~~FIG. 6 illustrates one example of a recent entry queue modifying the BTB table hit detect logic. In FIG. 6 block 600 includes a BTB table array 610, a recent entry queue 620 and a Hit Detect Compare Logic 630 which provides a hit detect output on line 640.~~ FIG. 6 illustrates an example of a recent entry queue 620 modifying the ~~status of~~ the BTB status ~~table array~~ 610. Since the recent entry queue 620 is a substantially smaller subset of the BTB ~~table array~~ 610, the ability to search for branch/target pairs in the recent entry queue 620 is significantly faster than searching in the ~~far larger~~ BTB ~~table array~~ 610. In the cycle ~~, when~~ the BTB ~~table~~ 610 is being accessed for a given row based on the search index, the recent entry queue 620 can be compared to the hit detect ~~compare logic~~ criteria 630 in parallel. Hence, whenever a new search is started, during the cycle ~~when~~ a read is being performed from the BTB table ~~array~~ 610, the recent entry queue 620 is doing a compare ~~with the Hit Detect Compare Logic~~ 630 on its contents during the same cycle. The ability to do a hit detect 640, or search match, a cycle earlier improves the latency factor of the branch prediction logic for tight looping branches where the same branch is accessed repetitively and the BTB table ~~array~~ 610 by itself is unable to keep up because of the initial time required to access the BTB ~~table~~ array 610. ~~As stated above, the recent entry queue 620 maintains a depth up to the associativity of the BTB table array 610 whereby while the BTB is indexed, the recent entry queue positions are input to comparison logic compare equal comparators 430,431 in FIG. 4 and hit detect compare logic 630 in FIG. 6. The recent entry queue depth is searched in respect to a matching branch in parallel with searching of the BTB output, where the hit detect compare logic 630 supports the associativity of the BTB table array 610. In searching the BTB table array 610 for the next predicted branch, the search strategy uses a subset of the recent entry queue as a subset of the BTB, table array 610 and preferably fast indexes recently encountered branches.~~

[0038] ~~FIG. 7 illustrates one example of a state machine descriptor of a recent entry queue delaying decoding.~~ In FIG. 7 block 700 illustrates an example of a state machine description of a recent entry queue delaying decode ~~ing.~~ As shown, because the BTB ~~table~~ 610 is working asynchronously from the decoding ~~stage~~ 310 of instructions in the microprocessor pipeline 300 of FIG. 3, it is possible for the pipeline to decode ~~, with the decoding stage~~ 310, a branch which is predictable, but was not yet found by the BTB ~~table~~ 610. In such cases, the branch is deemed as a surprise branch 710 and ~~upon operation of the branch stage~~ resolution ~~stage~~ 350, the execution cycle, of the branch, it will once again be placed into the BTB ~~table~~ 610. In the cases where this missed branch is a loop branch it will continue not to be predicted for after each occurrence of not finding it, the branch prediction logic will restart based on the surprise branch ~~decision block~~ 710. In the case that the branch is already in the BTB ~~table~~ 610 as detected ~~by OR gate~~ 440 by the recent entry queue 620, recent entry queue ~~decision block~~ 720, and the branch is a taken ~~as a~~ branch ~~from the resolved. In a decision block~~ 730 where the branch repeatedly occurs with a negative offset ~~, in branch point, decision block~~ 740, the recent entry queue 620 can be used to detect this scenario and cause the decoding 310 ~~of microprocessor pipeline 300~~ after the branch to be delayed ~~by the delay decode step~~ 760 until the first prediction via the BTB ~~table~~ 610 is made such that the predicted branch address can be compared against all future decodes ~~ing~~ ~~of the BTB table~~ 610. By causing a delay ~~in the delay decode step~~ 760 in the decode ~~ing~~ of the pipeline 300, the next iteration of the branch will be predicted by the BTB ~~table~~ 610 in time. Given that a BTB ~~table~~ 610 can have higher latency on start-up compared to the latency thereof once it is running, the one time delay of ~~by the decode in normal fashion step~~ 750, ~~or the delay decode block~~ 760 of decode ~~ing~~ can be enough to allow the branch to be predicted for all future iterations of the current looping pattern.

[0040] As one example, one or more aspects of the present invention can be included in an article of manufacture (e.g., one or more computer program products) having, for instance, a computer usable media having computer readable code thereon for controlling and configuring a computer machine having a pipelined processor and a branch target buffer (BTB) creating a recent entry queue in parallel with the branch target buffer (BTB). The computer usable media has embodied therein, for instance, computer readable program code means for providing and facilitating the capabilities of: the present invention. The article of manufacture can be included as a part of a computer system or sold separately.